EXP 1 : Consider a student database of SY AI class (at least 10 records). Database contains different fields of every student like Roll No, Name and SGPA.(array of objects of class). Design a roll call list, arrange list of students according to roll numbers in ascending order (Use Bubble Sort)

CODE :

```
#include <iostream>

#include <string>


using namespace std;


struct Student {

   int rollNo;

   string name;

   float sgpa;

};


void bubbleSort(Student arr[], int n) {

   for (int i = 0; i < n - 1; i++) {

       for (int j = 0; j < n - i - 1; j++) {

           if (arr[j].rollNo > arr[j + 1].rollNo) {

               swap(arr[j], arr[j + 1]);

           }

       }

   }

}


void displayStudents(Student arr[], int n) {

   cout << "Roll No\tName\tSGPA\n";

   for (int i = 0; i < n; i++) {

       cout << arr[i].rollNo << "\t" << arr[i].name << "\t" << arr[i].sgpa << "\n";

   }
```

```cpp
}

int main() {
    const int numStudents = 10;
    Student studentArray[numStudents] = {
        {101, "Alice", 3.5},
        {102, "Bob", 3.2},
        {103, "Charlie", 3.8},
        // Add more student records here
    };

    cout << "Before sorting:\n";
    displayStudents(studentArray, numStudents);

    bubbleSort(studentArray, numStudents);

    cout << "\nAfter sorting:\n";
    displayStudents(studentArray, numStudents);

    return 0;
}
```

EXP 2 : Consider a student database of SY AI class (at least 10 records). Database contains different fields of every student like Roll No, Name and SGPA.(array of objects of class). Arrange list of students alphabetically. (Use Insertion sort).

CODE :

```
#include <iostream>
#include <string>

using namespace std;

struct Student {
  int rollNo;
  string name;
  float sgpa;
};

void insertionSort(Student arr[], int n) {
  for (int i = 1; i < n; i++) {
    Student key = arr[i];
    int j = i - 1;

    while (j >= 0 && arr[j].name > key.name) {
      arr[j + 1] = arr[j];
      j = j - 1;
    }
    arr[j + 1] = key;
  }
}

void displayStudents(Student arr[], int n) {
  cout << "Roll No\tName\tSGPA\n";
```

```
    for (int i = 0; i < n; i++) {
        cout << arr[i].rollNo << "\t" << arr[i].name << "\t" << arr[i].sgpa << "\n";
    }
}

int main() {
    const int numStudents = 10;
    Student studentArray[numStudents] = {
        {101, "Alice", 3.5},
        {102, "Bob", 3.2},
        {103, "Charlie", 3.8},
        // Add more student records here
    };

    cout << "Before sorting:\n";
    displayStudents(studentArray, numStudents);

    insertionSort(studentArray, numStudents);

    cout << "\nAfter sorting alphabetically by name:\n";
    displayStudents(studentArray, numStudents);

    return 0;
}
```

EXP 3 : Consider a student database of SY AI class (at least 10 records). Database contains different fields of every student like Roll No, Name and SGPA.(array of objects of class). Arrange list of students to find out first ten toppers from a class. (Use Quick sort)

CODE :

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

struct Student {
    int rollNo;
    string name;
    float sgpa;
};

void searchBySGPA(Student arr[], int n, float targetSGPA) {
    vector<Student> matchingStudents;

    for (int i = 0; i < n; i++) {
        if (arr[i].sgpa == targetSGPA) {
            matchingStudents.push_back(arr[i]);
        }
    }

    if (matchingStudents.empty()) {
        cout << "No students found with SGPA " << targetSGPA << endl;
    } else {
        cout << "Students with SGPA " << targetSGPA << ":\n";
        cout << "Roll No\tName\tSGPA\n";
        for (const auto& student : matchingStudents) {
            cout << student.rollNo << "\t" << student.name << "\t" << student.sgpa << "\n";
        }
    }
}

int main() {
    const int numStudents = 10;
    Student studentArray[numStudents] = {
        {101, "Alice", 3.5},
        {102, "Bob", 3.2},
        {103, "Charlie", 3.5},
        // Add more student records here
    };

    float targetSGPA;
    cout << "Enter the SGPA to search for: ";
    cin >> targetSGPA;

    searchBySGPA(studentArray, numStudents, targetSGPA);

    return 0;
}
```

EXP 4 : Consider a student database of SY AI class (at least 10 records). Database contains different fields of every student like Roll No, Name and SGPA.(array of objects of class).Search students according to SGPA. If more than one student having same SGPA, then print list of all students having same SGPA.

CODE :

```
#include <iostream>
#include <string>

using namespace std;

class Student {
public:
    int rollNo;
    string name;
    float sgpa;
};

void searchBySGPA(Student students[], int numStudents, float targetSGPA) {
    bool found = false;

    cout << "Students with SGPA " << targetSGPA << ":\n";
    cout << "Roll No\tName\tSGPA\n";

    for (int i = 0; i < numStudents; i++) {
        if (students[i].sgpa == targetSGPA) {
            cout << students[i].rollNo << "\t" << students[i].name << "\t" << students[i].sgpa << "\n";
            found = true;
        }
    }
    if (!found) {
        cout << "No students found with SGPA " << targetSGPA << endl;
    }
}

int main() {
    const int numStudents = 10;
    Student studentDatabase[numStudents] = {
        {101, "Alice", 3.5},
        {102, "Bob", 3.2},
        {103, "Charlie", 3.5},
        {104, "David", 3.8},
        {105, "Emily", 3.2},
        {106, "Frank", 3.5},
        {107, "Grace", 3.8},
        {108, "Hank", 3.2},
        {109, "Ivy", 3.5},
        {110, "Jack", 3.8}
    };

    float targetSGPA;
    cout << "Enter the SGPA to search for: ";
    cin >> targetSGPA;

    searchBySGPA(studentDatabase, numStudents, targetSGPA);

    return 0;
}
```

EXP 5 : Consider a student database of SY AI class (at least 10 records). Database contains different fields of every student like Roll No, Name and SGPA.(array of objects of class). Search a particular student according to name using binary search without recursion.

CODE :

```
#include <iostream>

#include <string>


using namespace std;


struct Student {

   int rollNo;

   string name;

   float sgpa;

};


void bubbleSortByName(Student students[], int numStudents) {

   for (int i = 0; i < numStudents - 1; i++) {

      for (int j = 0; j < numStudents - i - 1; j++) {

         if (students[j].name > students[j + 1].name) {

            // Swap the elements if they are in the wrong order

            swap(students[j], students[j + 1]);

         }

      }

   }

}


int binarySearchByName(Student students[], int numStudents, const string& targetName) {

   int low = 0, high = numStudents - 1;


   while (low <= high) {
        int mid = low + (high - low) / 2;

        if (students[mid].name == targetName) {
            return mid;
        } else if (students[mid].name < targetName) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return -1;
}

void displayStudent(const Student& student) {
    cout << "Roll No: " << student.rollNo << "\n";
    cout << "Name: " << student.name << "\n";
    cout << "SGPA: " << student.sgpa << "\n";
}

int main() {
    const int numStudents = 10;
    Student studentDatabase[numStudents] = {
        {101, "Alice", 3.5},
        {102, "Bob", 3.2},
        {103, "Charlie", 3.8},
        // Add more student records here
    };

    bubbleSortByName(studentDatabase, numStudents);
    string targetName;
    cout << "Enter the name to search for: ";
    getline(cin, targetName);

    int index = binarySearchByName(studentDatabase, numStudents, targetName);

    if (index != -1) {
        cout << "Student found:\n";
        displayStudent(studentDatabase[index]);
    } else {
        cout << "Student not found.\n";
    }

    return 0;
}
```

EXP 6 : Department of Artificial Intelligence has student's club named ‘SAAI’. Students of Second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write program to maintain club member‘s information using singly linked list. Store student MIS registration no. and Name. Write functions to a) Add and delete the members as well as president or even secretary. b) Compute total number of members of club c) Display members

CODE :

```
#include <iostream>

#include <string>


using namespace std;


struct Node {

   int regNo;

   string name;

   Node* next;

};


Node* president = nullptr;

Node* secretary = nullptr;


void addMember(int regNo, const string& name) {

   Node* newNode = new Node{regNo, name, nullptr};


   if (!president) {

      president = newNode;

      secretary = newNode;

   } else {

      secretary->next = newNode;

      secretary = newNode;
```

```
    }

    cout << "Member added successfully.\n";
}

void deleteMember(int regNo) {
    Node* current = president;
    Node* previous = nullptr;

    while (current && current->regNo != regNo) {
        previous = current;
        current = current->next;
    }

    if (!current) {
        cout << "Member not found.\n";
        return;
    }

    if (current == president) {
        president = current->next;
    }

    if (current == secretary) {
        secretary = previous;
    }

    if (previous) {
        previous->next = current->next;
    }
    delete current;
    cout << "Member deleted successfully.\n";
}

int computeTotalMembers() {
    int count = 0;
    Node* current = president;

    while (current) {
        count++;
        current = current->next;
    }

    return count;
}

void displayMembers() {
    Node* current = president;

    cout << "Club Members:\n";
    while (current) {
        cout << "Reg No: " << current->regNo << "\tName: " << current->name << "\n";
        current = current->next;
    }
}

int main() {
    addMember(123, "John");
    addMember(456, "Alice");
    addMember(789, "Bob");
    displayMembers();

    deleteMember(456);

    displayMembers();

    cout << "Total members: " << computeTotalMembers() << endl;

    return 0;
}
```

EXP 7: Department of Artificial Intelligence has student's club named ‘SAAI’. Students of Second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write program to maintain club member‘s information using singly linked list. Store student MIS registration no. and Name. Write functions to Display list in reverse order using recursion

CODE : #include <iostream>

```
#include <string>

using namespace std;

struct Node {
    int regNo;
    string name;
    Node* next;
};

Node* president = nullptr;
Node* secretary = nullptr;

void addMember(int regNo, const string& name) {
    Node* newNode = new Node{regNo, name, nullptr};

    if (!president) {
        president = newNode;
        secretary = newNode;
    } else {
        secretary->next = newNode;
        secretary = newNode;
    }

    cout << "Member added successfully.\n";
}

void deleteMember(int regNo) {
    Node* current = president;
    Node* previous = nullptr;

    while (current && current->regNo != regNo) {
        previous = current;
        current = current->next;
    }

    if (!current) {
        cout << "Member not found.\n";
        return;
    }

    if (current == president) {
        president = current->next;
    }

    if (current == secretary) {
        secretary = previous;
    }

    if (previous) {
        previous->next = current->next;
    }

    delete current;
    cout << "Member deleted successfully.\n";
}
```

```
void displayList(Node* current) {
    cout << "Club Members:\n";
    while (current) {
        cout << "Reg No: " << current->regNo << "\tName: " << current->name << "\n";
        current = current->next;
    }
}

void displayReverseList(Node* current) {
    if (current) {
        displayReverseList(current->next);
        cout << "Reg No: " << current->regNo << "\tName: " << current->name << "\n";
    }
}

int main() {
    addMember(123, "John");
    addMember(456, "Alice");
    addMember(789, "Bob");

    cout << "Original List:\n";
    displayList(president);

    cout << "\nList in Reverse Order:\n";
    displayReverseList(president);

    return 0;
}
```

EXP 8 : Department of Artificial Intelligence has student's club named ‘SAAI’. Students of Second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write program to maintain club member‘s information using singly linked list. Store student MIS registration no. and Name. Make Two linked lists exists for two divisions. Concatenate two lists

CODE :

```
#include <iostream>
#include <string>

using namespace std;

struct Node {
    int regNo;
    string name;
    Node* next;
};

void addMember(Node*& head, int regNo, const string& name) {
    Node* newNode = new Node{regNo, name, nullptr};

    if (!head) {
        head = newNode;
    } else {
        Node* current = head;
        while (current->next) {
            current = current->next;
        }
        current->next = newNode;
    }
}
```

```
void deleteMember(Node*& head, int regNo) {
    if (!head) {
        cout << "List is empty. No member to delete.\n";
        return;
    }

    Node* current = head;
    Node* previous = nullptr;

    while (current && current->regNo != regNo) {
        previous = current;
        current = current->next;
    }

    if (!current) {
        cout << "Member not found.\n";
        return;
    }

    if (!previous) {
        head = current->next;
    } else {
        previous->next = current->next;
    }

    delete current;
    cout << "Member deleted successfully.\n";
}

void concatenateLists(Node*& firstList, Node*& secondList) {
    if (!firstList) {
        firstList = secondList;
    } else {
        Node* current = firstList;
        while (current->next) {
            current = current->next;
        }
        current->next = secondList;
    }

    secondList = nullptr;  // Set the second list to null after concatenation
}

void displayList(const Node* head) {
    cout << "Club Members:\n";
    while (head) {
        cout << "Reg No: " << head->regNo << "\tName: " << head->name << "\n";
        head = head->next;
    }
}

int main() {
    Node* division1List = nullptr;
    Node* division2List = nullptr;

    addMember(division1List, 101, "Alice");
    addMember(division1List, 102, "Bob");
    addMember(division1List, 103, "Charlie");

    addMember(division2List, 201, "David");
    addMember(division2List, 202, "Emma");
    addMember(division2List, 203, "Frank");


    cout << "Division 1 Members:\n";
    displayList(division1List);


    cout << "\nDivision 2 Members:\n";
    displayList(division2List);


    concatenateLists(division1List, division2List);


    cout << "\nConcatenated List:\n";
    displayList(division1List);


    deleteMember(division1List, 103);


    cout << "\nUpdated Concatenated List:\n";
    displayList(division1List);


    return 0;
}
```

EXP 9 : Implement Stack using a linked list. Use this stack to perform evaluation of a postfix expression

CODE : #include <iostream>

```
#include <stdlib.h>

#include <string.h>

#include <ctype.h>


struct node {

   int data;

   struct node *next;

};


struct node *top = NULL;


struct node *createNode(int data) {

   struct node *ptr = (struct node *)malloc(sizeof(struct node));

   ptr->data = data;

   ptr->next = NULL;

   return ptr;

}


void push(int data) {

   struct node *ptr = createNode(data);

   if (top == NULL) {

      top = ptr;

      return;

   }

   ptr->next = top;

   top = ptr;

}
```

```
int pop() {
    int data;
    struct node *temp;
    if (top == NULL)
        return -1;
    data = top->data;
    temp = top;
    top = top->next;
    free(temp);
    return (data);
}

int main() {
    char str[100];
    int i, data = -1, operand1, operand2, result;

    std::cout << "Enter your postfix expression: ";
    fgets(str, 100, stdin);

    for (i = 0; i < strlen(str); i++) {
        if (isdigit(str[i])) {
            data = (data == -1) ? 0 : data;
            data = (data * 10) + (str[i] - '0');
            continue;
        }

        if (data != -1) {
            push(data);
        }
        if (str[i] == '+' || str[i] == '-' || str[i] == '*' || str[i] == '/') {
            operand2 = pop();
            operand1 = pop();
            if (operand1 == -1 || operand2 == -1)
                break;

            switch (str[i]) {
                case '+':
                    result = operand1 + operand2;
                    push(result);
                    break;

                case '-':
                    result = operand1 - operand2;
                    push(result);
                    break;

                case '*':
                    result = operand1 * operand2;
                    push(result);
                    break;

                case '/':
                    result = operand1 / operand2;
                    push(result);
                    break;
            }
        }
        data = -1;
    }
```

```
    if (top != NULL && top->next == NULL)
        std::cout << "Output: " << top->data << std::endl;
    else
        std::cout << "You have entered a wrong expression." << std::endl;


    return 0;
}
```

EXP 10 : Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to add job and delete job from queue.

CODE :
```
#include <iostream>

#define MAX 10

using namespace std;


struct Queue {

   int data[MAX];

   int front, rear;

};


void initQueue(Queue &q) {

   q.front = q.rear = -1;

}


int isEmpty(Queue &q) {

   return (q.front == q.rear) ? 1 : 0;

}


int isFull(Queue &q) {

   return (q.rear == MAX - 1) ? 1 : 0;

}


void enqueue(Queue &q, int x) {

   q.data[++q.rear] = x;

}


int dequeue(Queue &q) {

   return q.data[++q.front];
```

```
}

void display(Queue &q) {
    int i;
    cout << "\n";
    for (i = q.front + 1; i <= q.rear; i++)
        cout << q.data[i] << " ";
}

int main() {
    Queue q;
    initQueue(q);

    int ch, x;
    do {
        cout << "\n 1. Insert job\n 2. Delete job\n 3. Display\n 4. Exit\n Enter your choice:";
        cin >> ch;

        switch (ch) {
            case 1:
                if (!isFull(q)) {
                    cout << "\n Enter data:";
                    cin >> x;
                    enqueue(q, x);
                } else {
                    cout << "Queue is overflow";
                }
                break;

            case 2:
                if (!isEmpty(q))
                    cout << "\n Deleted Element=" << dequeue(q);
                else
                    cout << "\n Queue is underflow";

                cout << "\nRemaining jobs :";
                display(q);
                break;

            case 3:
                if (!isEmpty(q)) {
                    cout << "\n Queue contains:";
                    display(q);
                } else {
                    cout << "\n Queue is empty";
                }
                break;

            case 4:
                cout << "\n Exit";
                break;
        }
    } while (ch != 4);

    return 0;
}
```

EXP 11 : A double-ended queue (deque) is a linear list in which additions and deletions may be made at either end. Obtain a data representation mapping a deque into a one-dimensional array. Write C++ program to simulate deque with functions to add and delete elements from either end of the deque

CODE :

```
#include <iostream>
using namespace std;

const int MAX_SIZE = 100;
struct Deque {
  int arr[MAX_SIZE];
  int front, rear;
};

void initDeque(Deque &dq) {
  dq.front = -1;
  dq.rear = 0;
}

bool isEmpty(Deque &dq) {
  return dq.front == -1;
}

bool isFull(Deque &dq) {
  return (dq.front == 0 && dq.rear == MAX_SIZE - 1) || dq.front == dq.rear + 1;
}

void addFront(Deque &dq, int data) {
  if (isFull(dq)) {
    cout << "Deque Overflow!" << endl;
```

```
        return;
    }

    if (dq.front == -1) {
        dq.front = 0;
        dq.rear = 0;
    } else if (dq.front == 0) {
        dq.front = MAX_SIZE - 1;
    } else {
        dq.front--;
    }

    dq.arr[dq.front] = data;
    cout << "Element " << data << " added at front." << endl;
}

void addRear(Deque &dq, int data) {
    if (isFull(dq)) {
        cout << "Deque Overflow!" << endl;
        return;
    }

    if (dq.front == -1) {
        dq.front = 0;
        dq.rear = 0;
    } else if (dq.rear == MAX_SIZE - 1) {
        dq.rear = 0;
    } else {
        dq.rear++;
    }
    dq.arr[dq.rear] = data;
    cout << "Element " << data << " added at rear." << endl;
}

void deleteFront(Deque &dq) {
    if (isEmpty(dq)) {
        cout << "Deque Underflow!" << endl;
        return;
    }

    if (dq.front == dq.rear) {
        dq.front = -1;
        dq.rear = 0;
    } else if (dq.front == MAX_SIZE - 1) {
        dq.front = 0;
    } else {
        dq.front++;
    }

    cout << "Element deleted from front." << endl;
}

void deleteRear(Deque &dq) {
    if (isEmpty(dq)) {
        cout << "Deque Underflow!" << endl;
        return;
    }

    if (dq.front == dq.rear) {
        dq.front = -1;
        dq.rear = 0;
    } else if (dq.rear == 0) {
        dq.rear = MAX_SIZE - 1;
    } else {
        dq.rear--;
    }

    cout << "Element deleted from rear." << endl;
}

void displayDeque(Deque &dq) {
    if (isEmpty(dq)) {
        cout << "Deque is empty." << endl;
        return;
    }

    int i = dq.front;
    cout << "Deque elements: ";
    while (i != dq.rear) {
        cout << dq.arr[i] << " ";
        i = (i + 1) % MAX_SIZE;
    }
    cout << dq.arr[dq.rear] << endl;
}

int main() {
    Deque dq;
    initDeque(dq);

    addFront(dq, 10);
    addRear(dq, 20);
    addFront(dq, 30);
    displayDeque(dq);

    deleteFront(dq);
    deleteRear(dq);

    displayDeque(dq);

    return 0;
}
```

EXP 12 : A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.

CODE :

```
#include <iostream>
#include <vector>
using namespace std;

struct Node {
   string label;
   vector<Node*> children;

   Node(string lbl) {
      label = lbl;
   }
};

void printNodes(Node* root) {
   if (root == nullptr) {
      return;
   }

   cout << root->label << endl;
   for (Node* child : root->children) {
      printNodes(child);
   }
}

int main() {
   Node* book = new Node("Book");
```

```
Node* chapter1 = new Node("Chapter 1");
Node* chapter2 = new Node("Chapter 2");

Node* section1_1 = new Node("Section 1.1");
Node* section1_2 = new Node("Section 1.2");
Node* section2_1 = new Node("Section 2.1");

Node* subsection1_1_1 = new Node("Subsection 1.1.1");
Node* subsection1_1_2 = new Node("Subsection 1.1.2");
Node* subsection1_2_1 = new Node("Subsection 1.2.1");

book->children.push_back(chapter1);
book->children.push_back(chapter2);

chapter1->children.push_back(section1_1);
chapter1->children.push_back(section1_2);
chapter2->children.push_back(section2_1);

section1_1->children.push_back(subsection1_1_1);
section1_1->children.push_back(subsection1_1_2);
section1_2->children.push_back(subsection1_2_1);

cout << "Nodes of the Book Tree:" << endl;
printNodes(book);

delete book;
delete chapter1;
delete chapter2;
delete section1_1;
delete section1_2;
```

```
    delete section2_1;
    delete subsection1_1_1;
    delete subsection1_1_2;
    delete subsection1_2_1;

    return 0;
}
```

EXP 13 : Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree i. Insert new node and display.

CODE :

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};

Node* insert(Node* root, int val) {
    if (root == nullptr) {
        return new Node(val);
    }

    if (val < root->data) {
        root->left = insert(root->left, val);
    } else {
        root->right = insert(root->right, val);
    }
    return root;
}

void display(Node* root) {
    if (root == nullptr) {
        return;
    }
    display(root->left);
    cout << root->data << " ";
    display(root->right);
}

int main() {
    Node* root = nullptr;

    int values[] = {50, 30, 70, 20, 40, 60, 80};
    for (int val : values) {
        root = insert(root, val);
    }

    cout << "Initial BST (inorder traversal): ";
    display(root);
    cout << endl;

    int newValue = 55;
    root = insert(root, newValue);

    cout << "BST after inserting " << newValue << " (inorder traversal): ";
    display(root);
    cout << endl;
    return 0;
}
```

EXP 14 : Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree ii. Find number of nodes in longest path and display.

CODE :

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};

Node* insert(Node* root, int val) {
    if (root == nullptr) {
        return new Node(val);
    }

    if (val < root->data) {
        root->left = insert(root->left, val);
    } else {
        root->right = insert(root->right, val);
    }
    return root;
}

int longestPath(Node* root) {
    if (root == nullptr) {
        return 0;
    }

    int leftDepth = longestPath(root->left);
    int rightDepth = longestPath(root->right);

    return 1 + max(leftDepth, rightDepth);
}

int main() {
    Node* root = nullptr;

    int values[] = {50, 30, 70, 20, 40, 60, 80};
    for (int val : values) {
        root = insert(root, val);
    }

    int longest = longestPath(root);
    cout << "Number of nodes in the longest path: " << longest << endl;

    return 0;
}
```

EXP 15 : Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree iii. Minimum data value found in the tree.

CODE :

```
#include <iostream>

using namespace std;

struct TreeNode {
  int data;
  TreeNode* left;
  TreeNode* right;

  TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};

TreeNode* insert(TreeNode* root, int value) {
  if (root == nullptr) {
    return new TreeNode(value);
  }

  if (value < root->data) {
    root->left = insert(root->left, value);
  } else {
    root->right = insert(root->right, value);
  }

  return root;
}
```

```
int findMin(TreeNode* root) {
    if (root == nullptr) {
        cerr << "Error: Tree is empty." << endl;
        exit(EXIT_FAILURE);
    }

    while (root->left != nullptr) {
        root = root->left;
    }

    return root->data;
}

void inOrderTraversal(TreeNode* root) {
    if (root != nullptr) {
        inOrderTraversal(root->left);
        cout << root->data << " ";
        inOrderTraversal(root->right);
    }
}

int main() {
    TreeNode* root = nullptr;

    int values[] = {5, 3, 8, 2, 4, 7, 9};
    int size = sizeof(values) / sizeof(values[0]);

    for (int i = 0; i < size; ++i) {
        root = insert(root, values[i]);
    }
    cout << "In-order traversal of the BST: ";
    inOrderTraversal(root);
    cout << endl;

    int minValue = findMin(root);
    cout << "Minimum value in the BST: " << minValue << endl;

    return 0;
}
```

EXP 16 : Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree iv. Change a tree so that the roles of the left and right pointers are swapped at every node.

CODE :

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};

Node* insert(Node* root, int val) {
    if (root == nullptr) {
        return new Node(val);
    }

    if (val < root->data) {
        root->left = insert(root->left, val);
    } else {
        root->right = insert(root->right, val);
    }
    return root;
}

void swapLeftRight(Node* root) {
    if (root == nullptr) {
        return;
    }

    Node* temp = root->left;
    root->left = root->right;
    root->right = temp;

    swapLeftRight(root->left);
    swapLeftRight(root->right);
}

void displayTree(Node* root) {
    if (root == nullptr) {
        return;
    }

    cout << root->data << " ";
    displayTree(root->left);
    displayTree(root->right);
}

int main() {
    Node* root = nullptr;

    int values[] = {50, 30, 70, 20, 40, 60, 80};
    for (int val : values) {
        root = insert(root, val);
    }

    swapLeftRight(root);

    cout << "Modified BST after swapping left and right pointers:\n";
    displayTree(root);
    cout << endl;

    return 0;
}
```

EXP 17 : Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree v. Search a value

CODE :

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;

    Node(int val) {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};

Node* insert(Node* root, int val) {
    if (root == nullptr) {
        return new Node(val);
    }

    if (val < root->data) {
        root->left = insert(root->left, val);
    } else {
        root->right = insert(root->right, val);
    }
    return root;
}

bool search(Node* root, int val) {
    if (root == nullptr) {
        return false;
    }

    if (root->data == val) {
        return true;
    } else if (val < root->data) {
        return search(root->left, val);
    } else {
        return search(root->right, val);
    }
}

int main() {
    Node* root = nullptr;

    int values[] = {50, 30, 70, 20, 40, 60, 80};
    for (int val : values) {
        root = insert(root, val);
    }

    int searchValue = 40;

    if (search(root, searchValue)) {
        cout << "Value " << searchValue << " found in the BST.\n";
    } else {
        cout << "Value " << searchValue << " not found in the BST.\n";
    }

    return 0;
}
```

EXP 18 : Implement graph using adjacency list or matrix and perform DFS or BFS

CODE :

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

struct Graph {
   int V; // Number of vertices
   vector<vector<int>> adjList;

   Graph(int vertices) {
      V = vertices;
      adjList.resize(V);
   }

   void addEdge(int u, int v) {
      adjList[u].push_back(v);
      adjList[v].push_back(u); // For undirected graph
   }

   void DFSUtil(int v, vector<bool>& visited) {
      visited[v] = true;
      cout << v << " ";

      for (int i = 0; i < adjList[v].size(); ++i) {
         int adjacentVertex = adjList[v][i];
         if (!visited[adjacentVertex]) {
            DFSUtil(adjacentVertex, visited);
            }
        }
    }

    void DFS(int startVertex) {
        vector<bool> visited(V, false);
        cout << "DFS Traversal starting from vertex " << startVertex << ":\n";
        DFSUtil(startVertex, visited);
        cout << endl;
    }
};

int main() {
    int vertices = 5;
    Graph g(vertices);

    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(2, 4);
    g.addEdge(3, 4);

    int startVertex = 0; // Starting vertex for DFS traversal
    g.DFS(startVertex);

    return 0;
}
```